# Objective

You will learn how to display bitmap (aka *raster*) images using Processing. You will explore the PImage class and the methods available to manipulating the pixels of an image. You will also practice array traversal.

## Topics: bitmap graphics, instance methods and fields, arrays, color

# Instructions

In the sketch, you will find code in the **setup()** method that loads an image file from your data directory. You will load the image twice, and create two instances of the PImage class, called image1 and image2. Read over the documentation for PImage, and get an idea of the methods available. <http://processing.org/reference/PImage.html>

For this lab, you will add code to the existing method called:

void **process**(PImage img)

In this method, you will receive a PImage object as a parameter. You will access the individual pixels of the image using the color[] pixels instance field that every PImage object contains. This array of color variables contains the color of each pixel that makes up the image.

For this lab, you will traverse the contents of the pixels array, and create a horizontally flipped image. First, we need to discuss the way pixel data is stored in a PImage object. The pixels array is a single-dimensional array containing all the data in the image. Each horizontal row of pixels is placed into the array by Processing, from top to bottom. This is called *row major* ordering.

However, when we work with the pixels, it is helpful to think of the pixels in terms of their (x,y) position within the image. A common idiom for converting from an (x,y) position to an index within a one-dimensional array is:

index = yPosition \* imageWidth + xPosition

In the **process()** method, you will use nested loops and the idiom shown above to access the individual color values stored in the pixels array. Using the position of each pixel, you will swap pixels from opposite end of the same row. You will repeat that process, moving inward along the row, until you have swapped all the pixels. The result will be a horizontally flipped image.

## Algorithm Help

1. Start with a spot on the row
2. Calculate the opposite position (length of row – spot - 1)
3. Copy the value of spot to temp storage
4. Copy the endSpot’s value to position spot
5. Copy the value in temp to the endSpot position
6. Advance the position of spot and repeat

Here is a visualization of the algorithm for horizontal flipping of an image:

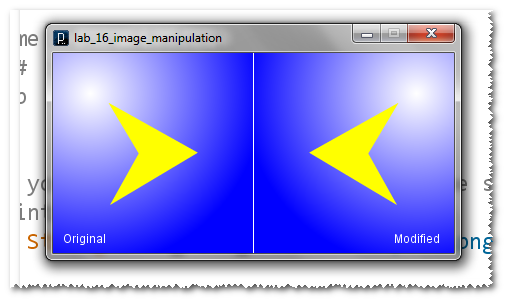
**First Iteration**

C:\Dropbox\Processing_Workshop\03 Object Oriented Programming Labs\lab_16_image_manipulation\Step 1.pngC:\Dropbox\Processing_Workshop\03 Object Oriented Programming Labs\lab_16_image_manipulation\Step 2.pngC:\Dropbox\Processing_Workshop\03 Object Oriented Programming Labs\lab_16_image_manipulation\Step 3.pngC:\Dropbox\Processing_Workshop\03 Object Oriented Programming Labs\lab_16_image_manipulation\Step 4.pngC:\Dropbox\Processing_Workshop\03 Object Oriented Programming Labs\lab_16_image_manipulation\Step 5.png

**Second Iteration**

C:\Dropbox\Processing_Workshop\03 Object Oriented Programming Labs\lab_16_image_manipulation\Step 6.pngC:\Dropbox\Processing_Workshop\03 Object Oriented Programming Labs\lab_16_image_manipulation\Step 7.pngC:\Dropbox\Processing_Workshop\03 Object Oriented Programming Labs\lab_16_image_manipulation\Step 8.pngC:\Dropbox\Processing_Workshop\03 Object Oriented Programming Labs\lab_16_image_manipulation\Step 9.pngC:\Dropbox\Processing_Workshop\03 Object Oriented Programming Labs\lab_16_image_manipulation\Step 10.png

# Examples



# Hints

If you iterate too far along the row, you will wind up swapping everything back into the original position! What should the condition be for your x loop?

# Challenge

Try adding code to the **process()** method to flip the image horizontally and vertically.

